PATENT
Docket No. 200314702-1
Express Mail Label No. EU790295319US

Computer Memory Initialization

Background

[0001] A computer system performs a power on self test (POST) procedure when it is initially turned on or reset in order to boot and configure itself for operation. POST is performed by a BIOS (Basic Input/Output System), or firmware, stored in a ROM (Read Only Memory) and executed by a processor within the computer system. Among other functions, POST includes initializing (configuring or programming) a memory subsystem of the computer system. The memory subsystem includes physical computer memory and a memory controller by which the physical computer memory is accessed. The physical computer memory includes one or more memory modules, such as SDRAM DIMMs (synchronized dynamic random access memory dual in-line memory modules). Each memory module includes an EEPROM (electrically erasable programmable ROM) or SPD (serial presence detect), which stores SPD data that identifies and describes the memory module. The EEPROM is a nonvolatile memory device, which can store data without losing the data during a system reset.

[0002] The initialization of the memory subsystem involves testing and sizing the computer memory (i.e. determining how much physical computer memory is present and useable) and configuring the memory controller to use the computer memory. To do so, the BIOS gathers type, size, speed and memory attributes from the EEPROMs of the memory modules and programs the memory controller accordingly.

[0003] Until the memory controller and memory modules are initialized, the memory subsystem cannot be used, so the computer system effectively has no useable memory. Once the memory controller has been configured, the BIOS will program a setting in the memory controller that triggers the memory controller to initialize the memory modules. Then the computer memory is capable of being used.

[0004] After the memory subsystem is initialized, it is possible to create, or initialize, variables and a software "stack" within the computer memory. The stack is commonly used in many programming functions, such as upon entering into and exiting from subroutines. Program code, or instructions, that use the stack are referred to as "stack-based." The instructions by which the BIOS initializes the memory subsystem, however, must be "stackless," since the stack cannot be created before the memory subsystem is initialized. Therefore, since the memory initialization instructions are stackless, among other limitations, they cannot include the convenient and simple "normal" method of using variables or using subroutines in which the contents of registers within the processor are saved to the stack upon entering a subroutine and restored to the registers upon exiting from the subroutine. Without the convenience of subroutines and variables, many of the instructions must be repeated within the firmware each time functions reoccur.

[0005] The stackless memory initialization instructions are limited to using the registers of the processor as the only available memory spaces. This limitation may not be significant for some processors that have over a hundred registers, but there are other processors that have a considerably smaller set of registers with which to work. The Opteron (TM) processor from AMD, for example, has only six registers. With so few registers, it is very important to be extremely careful how the registers are used. The stackless nature of the instructions operating on such processors, therefore, causes the memory initialization firmware to be relatively complex, detailed and lengthy to ensure that register contents are not lost during memory initialization. The complexity and length of the firmware, however, negatively impacts development cost and time incurred in generating and debugging the firmware.

[0006] For some computer systems that have multiple processors (multiple "nodes"), each processor is associated with its own memory subsystem. Upon reset, all of the memory subsystems within the computer system must be initialized. One of the processors, referred to as a "bootstrap" processor, performs all of the initializations, while the other processors remain temporarily inactive. The bootstrap processor surveys each node to discover each memory subsystem, reads all the SPD data from all the DIMMs for each node and programs the memory controller for each node. In this case, the stackless memory initialization instructions are even more complex, detailed and lengthy than in the single-processor case described above, since the instructions are not only stackless, but also must be able to handle multiple nodes.

Summary

[0007] According to a particular embodiment of the present invention, a method for performing computer memory initialization comprises generating configuration data for a portion of memory, saving the configuration data, restarting computer memory initialization, copying the saved configuration data to initialize the portion of memory, and using the portion of memory to execute instructions to initialize a remainder of memory.

[0008] According to another embodiment of the present invention, a method for initializing computer memory comprises resetting a computer system; determining whether the reset is firmware initiated; upon determining that the reset is firmware initiated, copying saved configuration data to initialize a portion of the computer memory; and using the portion of the computer memory to execute instructions to initialize a remainder of the computer memory.

[0009] Additionally, according to yet another embodiment, a computer system comprises first and second memory controllers, first and second computer memory associated with the first and second memory controllers, respectively, a nonvolatile memory space associated with the first computer memory, and firmware. Under control of the firmware the computer system generates configuration data that enables the first memory controller to use the first computer memory, saves the configuration data in the nonvolatile memory space, copies the configuration data to the first memory controller to initialize the first memory controller to use the first computer memory, and uses the first computer memory to initialize the second memory controller to use the second computer memory.

Brief Description of the Drawings

[0010] Fig. 1 is a block diagram of a computer system according to an embodiment of the present invention.

[0011] Fig. 2 is a block diagram of a memory subsystem according to an embodiment of the present invention and incorporated in the computer system shown in Fig. 1.

[0012] Fig. 3 is a block diagram of a ROM according to an embodiment of the present invention and incorporated in the computer system shown in Fig. 1.

[0013] Fig. 4 is a flow chart of a procedure according to an embodiment of the present invention for initializing memory in the computer system shown in Fig. 1.

Detailed Description

[0014] A computer system 100 incorporating an embodiment of the present invention is shown in Fig. 1. The computer system 100 generally includes one or more processors (e.g. Intel Pentium (TM), AMD Opteron (TM), etc.) 102, 104 and 106 connected to one or more memory subsystems 108, 110 and 112 through a bus system 114. The bus system 114 also preferably connects the processors 102-106 through one or more other bus systems 116 (e.g. a Peripheral Component Interconnect (PCI) bus, an Industry Standard Architecture (ISA) bus, etc.) to a ROM (read-only memory) 118. The processors 102-106 generally include various registers 119.

[0015] The memory subsystems 108-112, as shown in Fig. 2, preferably include memory controllers 122 and computer memory comprising one or more memory modules 124, 126 and 128. The memory modules 124-128 generally have one or more memory chips 130 and at least one EEPROM 132. The memory controllers 122 generally include various programmable registers 134.

[0016] The ROM 118, as shown in Fig. 3, generally stores firmware that includes, among other code, memory initialization code, or instructions, 136. The memory initialization code 136 further includes stackless instructions 138 and stack-based instructions 140.

[0017] Although an embodiment of the present invention is described with reference to the computer system 100, the invention is not limited to a computer architecture as shown and described, but may apply to any appropriate computer system and architecture. Additionally, although the memory subsystems 108-112 are shown to have identical configurations, the invention is not so limited, but may include any appropriate memory subsystems 108-112, each having any appropriate configuration. Furthermore, the computer memory may not all be in one sequential row, but may be distributed throughout the system, with cache coherency. Thus, each node is preferably cache coherent with the computer memory of the other nodes.

[0018] Upon a boot or a reset of the computer system 100, one of the processors 102-106 (e.g. the first processor 102) serves as a bootstrap processor and reads the contents of the ROM 118 in order to boot up the computer system 100. The memory initialization code 136 within the ROM 118 contains the instructions 138 and 140 under control of which the processor 102 initializes the memory subsystems 108-112. When the processor 102 encounters the memory initialization code 136, the processor

102 initially executes the stackless instructions 138 to find a first occurrence of memory in any one node. For example, the stackless instructions 138 preferably cause the processor 102 to read data stored in the EEPROM 132 of the first memory module 124 of the first memory subsystem 108. Alternatively, the processor 102 reads the EEPROMs 132 of the first and second memory modules 124 and 126, depending on the size of the data path and the size of the memory modules 124-128 used by the computer system 100 (e.g. a 128-bit data path will require pairs of 64-bit memory modules to operate). The data from the first EEPROM 132 (or first two EEPROMs 132) provides information with which, under control of the stackless instructions 138, the processor 102 programs, configures or initializes the memory controller 122 of the first memory subsystem 108 to use the first memory module 124 (or first two memory modules 124 and 126). The processor 102 then instructs the configured memory controller 122 to initialize the first memory module 124 (or first two memory modules 124 and 126).

[0019] At this point, the computer system 100 has a "minimal" amount of useable computer memory. Among the final functions of the stackless instructions 138 at this point, is to initialize a stack in the first memory module 124 (or first two memory modules 124 and 126). The processor 102 can thus use the stack under the control of the stack-based instructions 140 to discover (and in some embodiments, also initialize) the remainder of the memory modules 126-128 in the first memory subsystem 108.

[0020] The stack-based instructions 140 enable the usage of variables and subroutines. Thus, the stack-based instructions 140 may be considerably shorter and simpler than the stackless instructions of the prior art, since the stack-based instructions 140 can reuse the same subroutines and variables whenever needed, instead of repeating oft-used instructions at many locations within the memory initialization code 136. Since the amount and complexity of the stack-based instructions 140 is reduced, the time and cost to develop and debug the memory initialization code 136 is reduced.

[0021] Some currently available memory controllers 122 ("multi-init memory controllers") allow for more than one memory initialization operation after the same reset, but other currently available memory controllers 122 ("single-init memory controllers") allow for only one memory initialization operation after a reset. For a memory subsystem 108 having a multi-init memory controller, therefore, the stack-based instructions 140 are used to discover the remainder of the memory modules 126-128 in the first memory subsystem 108. Then the processor 102 returns to the stackless instructions (so as not to use memory while it is being initialized) to program the memory controller 122 with the complete data for using the entire set of memory modules 124-128, including any preferred memory interleaving scheme. The memory controller 122 is then instructed to initialize all of the memory modules 124-128. The memory subsystem 108 (with the multi-init memory controller) is thus fully configured and operational for use by the processor 102.

[0022] On the other hand, for a memory subsystem 108 having a single-init memory controller, upon discovering the remainder of the memory modules 126-128 in the first memory subsystem 108, the memory controller 122 cannot be programmed with the complete data for using the entire set of memory modules 124-128, because the memory controller 122 cannot yet initialize the remainder of the memory modules 126-128. Therefore, the complete configuration information is formatted and assembled, but not yet programmed into the memory controller 122. In this manner, the stack-based instructions 140 create the configuration information with "virtual" registers having a one-to-one correspondence with the actual registers that are in the memory controller 122. The configuration information is saved to the EEPROM 132 of the first memory module 124 (or two memory modules 124 and 126). Alternatively, the configuration information is saved to any available nonvolatile memory within the computer system 100. Since the memory to which the configuration information is saved is nonvolatile, the saved configuration information will survive a reset of the computer system 100. Therefore, at this point, a firmware-initiated reset is executed. Upon returning to the stackless instructions 138 after the firmware-initiated reset, the stackless instructions 138 copy the configuration information (i.e. the virtual registers) from the EEPROM 132 of the first memory module 124 (or two memory modules 124 and 126) to the registers 134 of the memory controller 122. The memory controller 122 is then instructed to initialize all of the memory modules 124-128. The memory subsystem 108 (with the single-init memory controller) is thus fully configured and operational for use by the processor 102.

[0023] Once the memory subsystem 108 is fully operational, whether after reset of the computer system 100 or after re-initialization of the memory modules 124-128, another stack is initialized in the memory modules 124-128. The stack-based instructions 140 then take over for discovery of the memory modules 124-128 and configuration of the memory controllers 122 in the other memory subsystems 110 and 112 and complete initialization of the other memory subsyst ms 110 and 112. In this manner, using the shorter and simpler stack-based instructions 140, the remaining memory subsystems 110 and 112 are made fully operational for use by the other processors 104 and 106.

[0024] In accordance with the above description, an embodiment of a procedure 142 for initializing the memory subsystems 108-112 during a boot process of the computer system 100 is described with reference to Fig. 4. Upon starting (at 144), it is determined (at 146) whether the current reset was initiated by firmware. This determination is preferably made by checking the value of a "firmware reset" flag (e.g. a particular bit in the EEPROM 132 of the first memory module 124 of the first memory subsystem 108). If the current reset is not firmware initiated, as determined at 146, then a "minimal" stackless memory initialization code of the stackless instructions 138 is executed (at 148) to discover the initial memory module 124 and EEPROM 132 connected to the memory controller 122 in the initial memory subsystem 108. Thus, the data from one of the EEPROMs 132 is read, one of the memory controllers 122 is programmed and at least one of the memory modules 124 is initialized, preferably the first occurrence of each. The software stack and any desired variables are then initialized (at 150) in the initialized memory module 124 for the programmed memory controller 122. The full stack-based memory initialization code can then be executed (starting at 152) to discover all of the computer memory for the programmed memory controller 122. The complete configuration information for the programmed memory controller 122 is generated and formatted (at 154) using the stack-based instructions 140. The formatted memory controller configuration information is saved (at 156) to the EEPROM 132 of the initialized memory module 124. The firmware reset flag is set and a firmware initiated reset is executed (at 158).

[0025] The procedure 142 ends (at 160), but the reset restarts the boot process of the computer system 100. When the procedure 142 restarts (at 144) during the boot process, the firmware reset flag is checked (at 146) again. It is thus determined that the current reset was initiated by firmware, since the firmware reset flag was set prior to the execution of the reset (at 158). The formatted memory controller configuration information is copied (at 162) from the EEPROM 132 of the first memory module 124 of the first memory subsystem 108 to the memory controller 122 of the first memory subsystem 108 using the stackless instructions 138. In this manner, the memory controller 122 is fully programm d. Therefore, the memory controller 122 initializes (at 164) its memory modules 124-128. The software stack and any desired variables are initialized (at 166) in the memory modules 124-128 for the initial memory controller 122. The full stack-based instructions 140 (Fig. 3) of the memory initialization code 136 are executed (at 168) for the memory controllers 122 and memory modules 124-128 of the remaining memory subsystems 110-112 (Fig. 1). The remaining memory controllers 122 and memory modules 124-128 are thus programmed and initialized. The procedure 142 ends (at 160), so any remaining portions of the boot procedure can be executed.